Critique of Unconstrained Memoryless Modular Strategies in Continual Learning

Author Name

August 13, 2024

Abstract

Starting with a critical review of Elastic Weight Consolidation method (EWC), we argue that unconstrained memoryless modular strategies does not account for all dimensions of pollution which is an inherent limitation to solve continual learning. So why does the litterature seem to show positive results? The reason lies in the introduction of hyperparameters (HPs) that create trade-offs between the tasks. Therefore, we argue that EWC should not perform any better than any other approach that makes similar trade-offs. To verify this, we present naive baselines along with a more elaborated method that only make trade-offs and show that, with the same amount of hyperparameters, they achieve similar performance to EWC. Additionally, we study how performances are impacted by the amount of hyperparameters, from $O(1)$ to $O(n^2)$. However, with all these HPs, we are essentially overfitting on the benchmarks, which we demonstrate through the introduction of validation benchmarks.

1 Introduction

1.1 Continual learning

We, humans are able to learn a wide variety of tasks throughout our lives. We can learn to play the piano, to speak a new language, to ride a bike, to cook a new recipe, etc. We can even learn to perform several tasks at the same time, such as playing the piano and singing at the same time. This ability to learn a sequence of tasks without forgetting the previously learned tasks is called continual learning. Continual learning is a fundamental aspect of human intelligence, and it is a key challenge in artificial intelligence. Indeed, when an artificial model using backpropagation is assigned to learn a sequence of tasks, its final state after training on a task is percieved as an initial state to train on the next task, which will be overwritten during the training procedure on the new task, leading to catastrophic forgetting of the previous task. This is a major limitation of backpropagation on neural networks, and it is a key challenge in artificial intelligence. In this paper, we will discuss the problem of continual learning and we will introduce a new approach to continual learning that is based on the idea of growing neural networks.

In a very broad sense, one can define continual learning as

the problem of learning several tasks sequentially.

Although this assumption is sometimes relaxed in the literature [REFLA], the fact that tasks have to be learnt *sequentially* implies that once the training procedure on a task is done, its associated training data cannot be used again.

Continual learning comes in different flavors that vary depending on the working assumptions we make. Some of the most common frameworks include the following[REFLA].

In *Task Incremental Learning*[REFLA], the model is trained sequentially on a series of tasks, with each task involving a different set of data or objectives, such as sentiment analysis and then object classification within images. The model knows which task it is dealing with at any given time, and it often has task-specific components or outputs. This framework is very general and can be applied to a wide range of problems. Then comes *Domain Incremental Learning*[REFLA] framework, within which the model faces a sequence of tasks that involve the same objective but in different domains. During inference, the model is not informed which domain it is operating in and must generalize across these domains. Finally, *Class Incremental Learning*[REFLA] involves learning new classes sequentially, without task identifiers. The model is expected to classify data from any of the classes it has learned so far, even as new classes are introduced. This is often considered the most challenging form of continual learning, as the model must avoid catastrophic forgetting while continuously expanding its ability to recognize new classes.

Litterature also aknwoledges other frameworks such as *Lifelong Learning* [REFLA], and other classifications of these frameworks [REFLA].

1.2 Approaches to continual learning

The problem of continual learning has been heavily investigated in the past decade, leading to the development of various approaches to mitigate catastrophic forgetting. These approaches can be broadly categorized into regularization-based, replay-based and architectural.

Regularization-based approaches, which constrain the learning process through a regularization term embedded in the loss function, aim at preventing drastic changes to model parameters that are critical for previous tasks. Fundational methods to this category of approaches include Elastic Weight Consolidation (EWC) [REFLA], which estimates the importance of each parameter and penalizes changes accordingly, and Synaptic Intelligence (SI) [REFLA], which dynamically accumulates information about parameters importance throughout training of all the tasks.

Another important approach is *replay-based learning*, which involves storing or generating data from previous tasks to revisit during the training of new tasks. This helps maintain performance on older tasks by refreshing the model's memory of past data. Methods like Experience Replay [REFLA] store a small buffer of past samples which are forwarded to the model when training on new tasks, while Generative Replay [REFLA] uses a external generative model to recreate as many data from previous tasks as needed.

Finaly, *architectural approaches* modify the model's structure to accommodate new tasks while preserving knowledge from previous ones. Methods like Progressive Neural Networks (PNN) [REFLA] add new subnetworks for each task, allowing forward knowledge transfer from old tasks dedicated subnetworks, while keeping the new ones isolated. Conversely, Mixture of Expert Models [REFLA] leverage a gating system which is designed to identify and distribute tasks to expert subnetworks.

However, these categories do not constitute a proper partition of the litterature as they are not mutually exclusive. Approaches such as Architectural and Regularization 1 (AR1) [REFLA] fall both in regularization-based and architectural categories. Additionally, these categories do not encompass certain methods such as adversarial approaches [REFLA].

1.3 Our specific framework

Given the variety of frameworks and approaches, we need to narrow down and specify ours. In previous subsections, we introduced frameworks and approaches to continual learning through categories. But beyond these categories, which provide a useful overview of the literature, what truly matters are the specific assumptions under which we operate.

First, we will focus on classification tasks involving the same set of classes. We assume that the model is notified when transitioning to the training of a new task. Each task must be handled strictly sequentially, meaning we do not permit any use of data from previous tasks. Additionally, we assume the total number of tasks is unknown and cannot be leveraged during the learning process.

As for the methods, our approach will be *modular*, meaning it will break down the learning process into subproblems and explicitly leverage dedicated modules to address these subproblems. These modules may consist of independent subnetworks, subnetworks that share some parameters, or groups of neurons distributed within the entirer network.

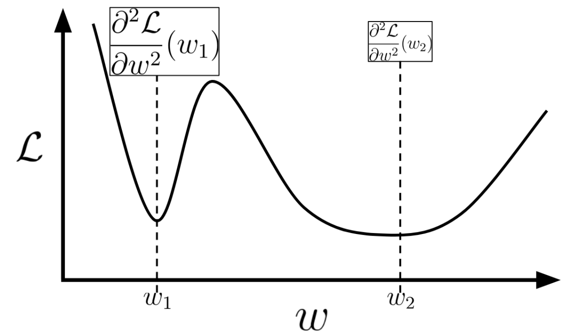


Figure 1: Illustration of our sense of importance of weights. When the second derivative of the loss with respect to a weight is high, the weight is considered important as modifying even a little bit would lead to a significant change in the loss. However, when the second derivative is low, modify the weight has little impact on the loss.

On the top of the modularity of our approach, we make to additional assumptions regarding the way we will process information. First, we will work with *unconstrained* architectures, meaning that we will not impose any architectural constraint on the model that could allow the processing or implicit remembering of the specificities of the past task. To this regard, convolutional or graph neural networks are out of our scope. We will also work with *memoryless* strategies, meaning that in addition to banning the use of data from previous tasks, we will not allow to leverage a replay buffer or generative rehearsal module.

2 Our approach : GroHess

2.1 Motivation

To address the problem of Continual Learning, one approach is to train independent specialized modules and a gating system that learns to recognize tasks, similar to Mixture of Experts Models [REFLA]. However, the tasks to be learned often have redundant features that we would like to leverage, so that only the new features of a task are learned without forgetting the old ones. The emergence of such features relies on constraints, starting with imposing small size of the model. Therefore, we would prefer to start training with a model of moderate size and expand it as more tasks are introduced.

Une telle approche présente plusieurs challenges. D'abord, il faut déterminer quelles sont les caractéristiques importantes à conserver. Ensuite, il faut déterminer comment les conserver. Enfin, il faut déterminer comment les utiliser pour traiter les nouvelles tâches. - Et l'objectif quand on ajoute des neurones à notre architecture est alors de savoir lesquels sont important et lesquels ne le sont pas. Pour cela, on introduit GroHess, une nouvelle approche de croissance d'architecture dans le cadre du continual learning.

2.2 Description

We control the growth of the model through two percentiles : the gradient percentile and the Hessian percentile. These percentiles allow us to define what is considered to be "high", relatively to the other weights first and second derivative. When the gradient of a given weight falls above the gradient percentile, comparatively to the other gradients, it means that according to backpropagation, this weight has to be strongly updated. Conversely, when the second derivative of a given weight with respect to the loss falls above the hessian percentile, comparatively to the second derivate of the other weights, it means that modifying this weight will strongly disturb the what has been learnt on the task. As a consequence, when both the first and the second derivatives of a weight are high, we grow a new neuron, to accodmodate both backpropagation and the preservation of the knowledge of the task. See OOOOOOOOOOOOOOOOOOO for illustration.

Algorithm 1 Training GroHess on n tasks

Input:
- A list of n classification tasks $L = [(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)]$
Output: A model ϕ able to solve each task

Step 1: Initialize the core model ϕ and train it on (D_1, L_1)
for each task (D_i, L_i) in the list **do**
  Step 2: Freeze parameters of ϕ and initialize LoRA parameters $\mathbb{L}_i$.
  Step 3: Train LoRA parameters on task (D_i, L_i) using ϕ for forward propagation.
  Step 4: Update ϕ by adding $\mathbb{L}_i$ parameters.
end for
Step 5: The core model ϕ is now trained on each task

2.3 Baselines

Our method is based on two ideas : a hessian-inspired mechanism to define weights importance and a growth mechanism to add new neurons to the model. To evaluate the relevance of these ideas, we introduce several baselines.

To evaluate the effectiveness of our method, which incorporates a Hessian-inspired mechanism for determining weight importance and a growth mechanism for adding new neurons to the model, we introduce four baselines.

In a first "vanilla" baseline, we define a model in the same initial configuration as the one used optimized with GroHess. During the training procedure through the tasks, no additional neurons are added, and no special mechanism is used to adjust or freeze weights according to their importance for the tasks. The model remains at its original size throughout the training process.

We also train a second "vanilla" baseline, but the initial size of its hidden layers is that of the average final hidden layer sizes when training with GroHess. So, from the beginning, this baseline is in the average configuration of a model trained with GroHess.

We also design a third baseline, where perform growth without GroHess, during the training procedure. This baseline involves starting the training process on the first task with a model of the same size as the one used with GroHess. However, everytime the model moves to training on a new task, it blindly adds new neurons to each layer, randomly initialezed, and without freezing any weight. This baseline helps evaluate the benefits of our approach of defining weights importance through Hessian coefficients.

Finally, our fourth and last baseline uses GroHess to freeze important weights without triggering the growing mechanism.This baseline is initialized in the same way as the second one, however, we leverage GroHess to freeze important weights during the training process, but without growing new neurons as GroHess would normaly do. This baseline is supposed to isolate the effectiveness of the growing mechanism to improve perfomances.

These baselines allow us to thoroughly analyze the contribution of the two main mechanisms of our approach, namely the Hessian-inspired weight importance mechanism and the growth mechanism, independently and in combination.

3 Main experiment

3.1 Permuted MNIST

A permuted MNIST (p-MNIST) dataset [REFLA] is a variant of the classic MNIST dataset where images are transformed through a fixed random permutation of pixel positions, creating a shuffled version of each original image. The label associated with each image remains the same, meaning the digits themselves are unchanged, but their visual structure is modified. Then, a p-MNIST task is define as solving the classification problem associated to a p-MNIST dataset. As each p-MNIST dataset is generated by a different permutation of MNIST dataset, there are 784! possible p-MNIST datasets and as many p-MNIST tasks. Finally, a p-MNIST benchmark is created by selecting a subset of these tasks. See 2 for illustration.

Permuted MNIST is particularly significant in the context of continual learning as it offers the opportunity to generate arbitrary long sequences of tasks with a fixed number of classes and a fixed number of samples per class. Additionally, users have a great control over the difficulty of a benchmark, which is directly associated with the similarity of the permutations used to generate the tasks of the benchmark. Finally, the p-MNIST benchmark is a well-established benchmark in the continual learning literature, which allows for comparison of different approaches.

On the other hand, litterature has come with a lot of criticism about the p-MNIST benchmarks. First, they are rather simple benchmarks, which do not reflect the complexity of real-world tasks. Second, the random permutations break the spatial structure of images we can find

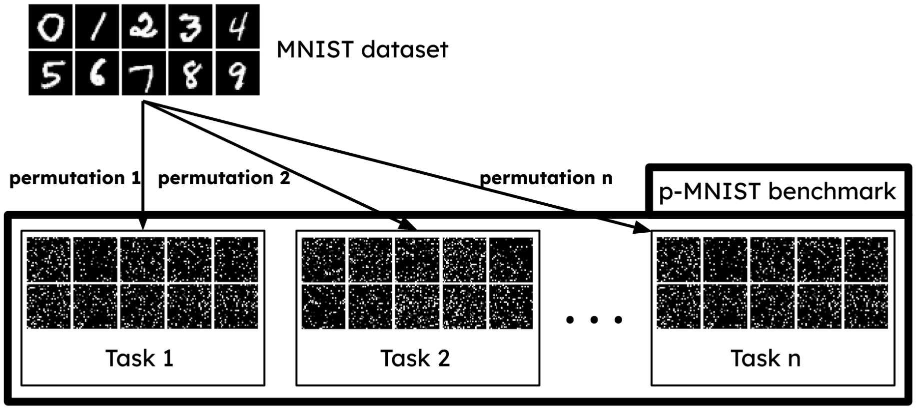


Figure 2: Illustration of the process of creating a p-MNIST benchmark.

in the real world, which means that the p-MNIST benchmarks are not well-suited to evaluate method that require spatial reasoning on tasks such as object detection or segmentation. In particular, the permutations of pixels disrupt with the assumption of locality and translation invariance which are at the basis of convolutional neural networks. As a consequence, approaches leveraging convolutional neural networks are disadvantaged on p-MNIST benchmarks.

Despite this criticism, we have chosen to use p-MNIST benchmarks. They are relevant for our study of GroHess as their very simplicity allows to understand the fundamental principles of our exploratory method. Additionally, the ability to generate not only several tasks, but also several comparable benchmarks will allow us to motivate and introduce a validation paradigm in continual learning.

Hyperparameters (HPs)	
Fixed HPs	**Value**
Number of Layers	3
Initial hidden layers size	300
Batch Size	128
Optimizer	Adam
Loss	Cross-entropy
Growth happens from	Output
Gradient percentile	0.98
Hessian percentile	0.98
Optimized HPs	**Range**
Learning Rate	1e-5 - 2e-3
Number of epochs	2 - 10

Table 1: Hyperparameters value or range

3.2 Training protocol

We train our models following GroHess algorithm on a sequence of 10 p-MNIST tasks, each with the usual 10 MNISTS classes. For each task 80% of the train set is used for training, 20% is used for validation and 100% of the test set is used to test the model. We train GroHess the Adam optimizer and the cross-entropy loss. For each task, the number of epochs and the learning rate are left as optimizable hyperparameters. We use a batch size of 128. See Table 1 for the complete list of hyperparameters.

Our model consist in a multi layer perceptron with 2 hidden layers (or 3 layers in total). Input and output sizes are 784 and 10 respectively. Each hidden layer contains initialy contains 300 neurons. The growth of the model happens from the output layer, which means that when a weight has both a high first and second derivative, we grow a neuron on the layer preceeding it (in the forward sense). We control the growth of the model through two percentiles : the gradient percentile and the Hessian percentile. When both first and second derivative of the loss with respect to a weight are above the gradient and Hessian percentiles, respectively, we grow a new neuron.

is considered important and a new neuron is added to the output layer.

4 Results

We evaluate the model on each task after training, and we report the average accuracy over all tasks. We repeat this process 10 times with different random seeds and we report the average accuracy over all runs.

5 Critical comparison to litterature

5.1 Geometrical argument

5.2 Combinatorial argument

6 Paradigmatic discussion

6.1 Validation

6.2 Hyperparameters optimizations

7 Additional remarks and opening

8 Conclusion

9 Our criticism

9.1 Pollutions

As a very general principle, the problem with these approaches in the context of continual learning is that

Improving on task 2 means deteriorating on task 1, and reciprocally.

It does not mean that the more tasks there are to solve, the harder it gets to solve them all, which is true. But this statement is stronger, it means that tasks are inhrently put in a competition setting with each other. This implies that performing continual learning in such a way requires to perform trade-offs. And the most convenient way one performs trade-offs in deep learning is through hyperparameters.

Let's assume we are given I tasks indexed in increasing order by the set $[\![1, I]\!]$. We define a neural network as the set of its parameters

$$\phi = \{\{w_{l,m,n} \in \mathbb{R} | m, n \in [\![1, N_{l-1}]\!] \times [\![1, N_l]\!]\} \,|\, l \in [\![1, L+1]\!]\} \tag{1}$$

where L is the number of hidden layers and N_l is the number of neurons on layer l.

Let's define $\mathbb{W}^i_{l,n}$ to be the set of protected weights for task i with respect to neuron n of layer l, for any $i \in [\![1, I]\!]$, $l \in [\![1, L+1]\!]$ and $n \in [\![1, N_l]\!]$:

$$\mathbb{W}^i_{l,n} = \{w_{l,m,n}, m \in [\![1, N_{l-1}]\!] | w_{l,m,n} \text{ protected}\} \tag{2}$$

which allows to define the pollution of task i by protected weights for task j on neuron n of layer l as

$$p^{i\to j}_{l,n} = \mathbb{E}_{x\sim D_i}\left[\frac{\sum_{w\in\mathbb{W}^j_{l,n}} w(x)}{\sum_{w\in\mathbb{W}^i_{l,n}} w(x)}\right] \tag{3}$$

Finally, we can define the pollution of task i by protected weights for task j on the whole network as

$$P^{i\to j} = \frac{1}{L+1}\sum_{l=1}^{L+1}\frac{1}{N_l}\sum_{n=1}^{N_l} p^{i\to j}_{l,n} \tag{4}$$

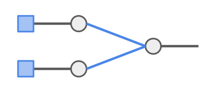
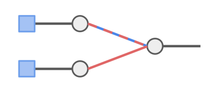
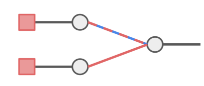
Trained / Tested	Taks 1	Task 2
Task 1		
Task 2		

Table 2: After training on task 1, there is no pollution (top left). After training on task 2, there is $P^{1\to 2}$, pollution from task-2-specific weights when forwarding task 1 samples (top right) and $P^{2\to 1}$, pollution from task-1-specific weights when forwarding task 2 samples (bottom right).

The notion of pollution $P^{i\to j}$ encapsulates how much neurons specialized for a task j interfere whith neurons specialized for a task i when provided a sample from task i, after training on the first $\max(i, j)$ tasks at least. See Table 2 for a simple illustration. Once the model is trained on task 1, we can test it on data from task 1 (top left cell) but we can't expect it to perform well on data from the next tasks (bottom left cell), so no pollution is occuring. Once the model is trained on task 2, we expect it to perform well on task 1 and 2, so the pollutions that can occur are $P^{2\to 1}$ (top right cell) and $P^{1\to 2}$ (bottom right cell). $P^{2\to 1}$ is generally handled by regularization approaches that keep certain weights of the model close to the weights of the model after training on task 1. The training process on task 2 take into consideration these dedicated weights, and the model learns to forward samples from task 2 through all the weights. However, $P^{1\to 2}$ is not handled by these approaches, as the model can't learn how to forward samples from task 1 through the undedicated weights retrained on task 2.

More boradly, the challenge is to deal with $P^{i\to j}$ for $j \neq i$, which requires to learn data from task i without being polluted by data from task j. Regularization approaches such as EWC claim to solve pollution of task j over task i ($P^{i\to j} = 0$) for $j < i$. While they can't really overcome it, they can at least mitigate it by penalizing the distance between the weights of the model after training on task i and the weights of the model after training on task j. However, they do not tackle the minimization of $P^{i\to j}$ for $j > i$. But they have a good reason no to tackle it : this is a fundamental limitation of this familly approaches, as it is inherently not possible to deal with it. Indeed, without contraints or some form of memory of the previous tasks, when provided a sample from task i, the model has no discrimination power of weights updated for task j from the ones the approach identified as specific for task i when training on task j, for $j > i$. So it can't know how to react to a sample from task i after training on task $j > i$.

Conversely, allowing some form of memorization of previous data distribution may enable to discriminate between samples features. This is has been attempted through replay memory approaches [19], [18] and rehearsal methods [17].

Alternatively, it can also be achieved through gating mechanisms such as mixture of expert models [6], [3]. These approaches are not memoryless. However, in memoryless approaches we have no direct control over the way the neurons specialized to task j will react when fed with samples from task $i < j$. So the very last hopes without memory is that the weights are such that they implicitly encode information about the task's specificities. This can be performed with constrained models, such as CNN, through which one could enforce convolution kernels associated to one task to be orthogonal to the convolution kernels associated to another task. And with additional contraints, this would prevent the model from using the same features for two different tasks while selecting for the most relevant features for each task. But, we considere unconstrained approaches, so the model has no reason to encourage such behavior.

9.2 Hyperparameters (mis)manipulations in continual learning

Continual learning enables several ways to manipulate hyperparameters. In this subsection, we will discuss various approaches to manipulate them. The perspective of most papers is to introduce a hyperparameter at the task-level, generaly in the loss used to train on a task. That is, they write the loss used while training on task 1 and the loss used for training on task 2, where they introduce the hyperparameter. This hyperparameter appears as an attempt to control the trade-off between the importance of the new task and the importance of the old tasks. However, they remain fuzzy about what happen to this hyperparameter on the latter tasks and the way to choose the value for the hyperparameter.

Let's introduce a hyperparameter λ. One could set this hyperparameter to be the same for all the tasks, which would result in a total of 1 hyperparameter. Alternatively, one could set this hyperparameter to a specific value λ_i for each task $i > 1$, which would result in a total of $n - 1$ hyperparameters. Or one could even set a different value for this hyperparameter for each pair of tasks, meaning that one could introduce $\lambda_{i,j}$ for mitigating catastrophic forgetting of task $j < i$ when learning task $i > 1$. This would result in a total of $\frac{(n-1)n}{2}$ hyperparameters *i.e.* $O(n^2)$ hyperparameters. Thus, it appears that the fuzziness of authors about the precise number of hyperparameters is not a minor issue as it can lead to very different hyperparameters manipulation.

Not only is the number of hyperparameters introduced by the authors not clear, but also the way they are chosen. If they introduce only 1 hyperparameter, that is the same for all tasks, then in the best case, it is chosen by hand, which is a very bad practice. In this case, the authors should at least explain why they chose this value or provide a heuristic, which is often not the case. In the worst case, they choose this hyperparameter through a hyperparameter optimization over all the tasks – see [Table 3] top-left – which is highly problematic due to data leakage. This breaks the continual learning framework as, after deployement, one has to decide the value of the hyperparameter when training on task 2 without accessing data from the latter tasks to help deciding. This is considered cheating in the context of continual learning. However, it seems that several papers are doing so. The same could be performed with in the $O(n)$ and $O(n^2)$ hyperparameters scenarios – see [Table 3] top-middle and top-right – which would be even more problematic, but as it looks even more obviously problematic, it seems that no one had the idea to do so.

In the three cases discussed above, we considered a single HPO performed on the training through all tasks simultaneously, which we regard as unfair. An alternative approach is to perform an HPO for each task individually, in a greedy fashion. This approach assumes at least one hyperparameter by task, so $O(1)$ hyperparameters is not suited – see [Table 3] bottom-left. This approach enables defining $O(n^2)$ hyperparameters $\lambda_{i,j}$ for mitigating catastrophic forgetting of task $j < i$ when learning task $i > 1$, where one performs an HPO on task i once the model is already trained on tasks $j < i$, before moving to task $i + 1$ – see [Table 3] bottom-right. However, we argue that it represents to many parameters, and makes questionnable the sacalability of this approach. Additionnaly, it is likely to lead to an overfitting on benchmark through over-specificity of tradeoffs. Finally the most reasonable approach seems to perform an HPO on each task individually, in a greedy fashion, but with only $O(n)$ hyperparameters, one for each task – see [Table 3] bottom-middle. This approach is compatible with the continual learning framework, as setting the hyperparameter for training on task i can be done without accessing data from the latter tasks.

Actually, what we refere to as "cheated HPO" where a single HPO is performed through all the tasks at the same time, could be just fine if one was performing the HPO on a benchmark, and then show the performances of the approach on another benchmark with the same hyperparameters. But this is not the case, as authors report performance on the benchmarks they performed hyperparameter optimization on. As we mentioned earlier, the role of these hyperparameters is to perform trade-offs between how much the model remembers of each task. So, *apriori*, we argue that there is no reason to believe that the hyperparameters that are good for a benchmark are good for another benchmark as they encapsulate the relative differences of the tasks within the benchmark. This benchmark-specificity of hyperparameters is the source of what we refere to as "overfitting on benchmark". Additionnaly, one should note that the cheated HPO implicitly exploit the number tasks, so the values of the hyperparameters obtained through this HPO would only be suited for benchmarks containing the same amout of tasks, which offers very narrow deployement opportunities.

A relevant inquiry would be to quantify this overfitting. An approch to do so is to perform HPO on a benchmark and then measure the drop of performance on another benchmark. Our hypothesis is that we should observe very little drop of performance between two benchmarks made of 10 different p-mnist task. However, if we decide of the hyperparameters on a benchmark made of 10 p-MNIST tasks and then test the strategy on a benchmark made of 10 soft 8×8-p-MNIST task, as defined in [9], we expect a drop in perfomances. This drop should be even more visible if test benchmark is derived from a different dataset, such as CIFAR-100. Note that, with the willingness to be very careful and precise, a proper quantification of overfitting of a strategy should be considered in relation to the similarity between benchmarks. Indeed, it is reasonnable that two very different benchmarks require two different sets to of hyperparameters. Unfortunately, this requires to define and quantify similarity between tasks, and it is a subtle inquiry to build a relevent metric that is versatile enough.

Finally, it appears that manipulation of hyperparameters in the context of continual learning is subtle, and we encourage more care in their management. Additionnaly, hyperparameters keep being thought at the scale of tasks, but instead of task-wise hyperparameters, a with thinking hyperparameters at the scale of the entire benchmark could be an interesting new perspective, more aligned with the continual learning framework. However, this would require to conducts HPOs on one or multiple benchmarks and evaluate the performance on a different benchmark using the same set of hyperparameters, to prevent overfitting on benchmarks.

9.3 An example of specious paper

A good example of paper that encapsulate all these issues :

- No code available. Even though the community reimplemented the method, the fact that atuhors do not publicly release the code is still an issue because when reproducing the code, one introduces the hyperparameters but cannot replicate the way have been manipulated by authors.
- Authors introduce the method and even explain it in greate details [1] for 2 tasks, but they don't discuss the challenges of scaling it to more tasks. AS a consequence, we don't know what is the loss used for training on latter tasks. They simply write, without more details

  > *"This can be enforced either with two separate penalties, or as one by noting that the sum of two quadratic penalties is itself a quadratic penalty"*

  which is ambiguous and insufficient to be clear about the number of hyperparameters introduced by their method over the entire continual learning process.
- In appendix, authors provide a table containing the value of hyperparameters. The table is quite exhaustive, except when it comes to λ, which is the hyperparameter they introduced through their method.
- As we demonstrated above, it is not enough to declare the values of the hyperparameters. However, when it comes to explain why or how these values were chosen, authors are fuzzy once again. They write

  > *When random hyperparameter search was used, 50 combinations of parameters were attempted for each number experiment*

  which does not enables to identify which of the 5 approaches to HPO they used, of if they used another approach.
- Despite this paper having 14 authors, it contains several inconsistencies in the name of its tables and figures. In section 4.1, about MNIST experiments, authors refere to Figure 3, which summarizes results of the reinforcement learning experiment. Conversely, Table 2, containing the value of hyperparameters for the reinforcement learning experiment, is entitled "Hyperparameters for each of the MNIST figures".

Authors of EWC provide a geometric intuition for their method that we reproduced in 3a, explaining that, after training on task A, the regularization prevent parameters from converging toward the argument of the minimum of the loss for task B when learning on task B. On the contrary, the parameters shoud converge to an overlapping region where both loss for task and loss for task B are small.

More precisely, the blue area represents a region of the parameter space where $\mathcal{L}_A$ is low - let's say, below a certain threshold. Similarly, the red area represents a region of the parameter space where $\mathcal{L}_B$ is low - let's say, below the same threshold.

However, the intuition suggested by this illustration is quite misleading and we argue that in practise, it doesn't hold.

First, no control over the distance between low loss regions for task A and task B. The illustration suggests that the low loss regions for task A and task B are close enough to overlap. But in practise, the distance between these regions is not controlled by the authors, and we have no reason to believe that the low loss regions for task A and task B are close enough to overlap. In fact, we have no reason to believe that the low loss regions for task A and task B overlap at all.

the area (or volume) of this overlapping region tends doubly towards 0.

On one hand, this representation accounts for two tasks, but in real life scenarios, there are many more. And as the number of tasks increases, the area of the overlapping region decreases, has 0 zero as a lower bound, so converges. But it has no reason to converge toward a non-zero value or at least, nobody has provided a better lower bound than 0 so far. On the other hand, In the illustration, the parameter space is $\mathbb{R}^2$. However, in deep learning, the parameter space is $\mathbb{R}^n$ with $n \sim 100000$ or even orders of magnitudes bigger. And the previous remark implies that there exist a number of tasks for which the overlapping region is small enough to fit within the Euclidean unit sphere, which volume converges toward 0 the dimension of the space increases.

These two remarks do not constitute a solid argument against the method as 3a is an illustrative depiction to build an intuitive understanding. But the intuition it provides

Number of HPOs \ Number of HPs	$O(1)$	$O(n)$	$O(n^2)$
1 cheated HPO	$T_1 \cdots T_m$ / $\{\lambda\}$ HPO	$T_1 \cdots T_m$ / $\{\lambda_1, ..., \lambda_n\}$ HPO	$T_1 \cdots T_m$ / $\{\lambda_{i,j} \mid \forall 2 \leq j < i \leq n\}$ HPO
n greedy HPOs		T_1 / $\{\lambda_1\}$ HPO_1 $\longrightarrow \ldots \longrightarrow$ T_n / $\{\lambda_n\}$ HPO_n	T_2 / $\{\lambda_{2,1}\}$ HPO_2 $\longrightarrow$ T_3 / $\{\lambda_{3,1}, \lambda_{3,2}\}$ HPO_3 $\longrightarrow \ldots \longrightarrow$ T_n / $\{\lambda_{n,1}, \ldots, \lambda_{n,n-1}\}$ HPO_n

Table 3: HPOs in continual learning

is misleading and does not reflect the complexity of the problem. Our remarks point out the absence of guaranty for the low loss regions to overlap. The absence of guaranty is not dramatic though, as deep learning tends to be able to perform without theoretical results in practice. But what is more problematic is that authors do not even have control over the overlapping, as they don't measure it.

In addition to this criticism, one should also note that the representation in 3a depicts a very optimistic case scenario. When training on task B the loss used by EWC can be written as $\mathcal{L} = \mathcal{L}_B + \lambda \mathcal{L}_{reg}$, where $\mathcal{L}_B$ accounts for the performance on task B, λ is a positive scalar and $\mathcal{L}_{reg}$ accounts for the regularisation. As introduced by EWC, $\mathcal{L}_{reg}$ aims at encouraging updates of parameters. Indeed, we identify two independant parameters that makes a case hard or easy : the orientation of the ellipse associated to low loss region for task B and the direction of the angle between the goal directions according to $\mathcal{L}_{reg}$ and $\mathcal{L}_B$. Indeed, in 4b, we see that the goal direction according to $\mathcal{L}_{reg}$ is aligned with the axis passing θ_A^* and the center of the overlap region, which is the most convenient scenario. By constrast, the low loss region associated to task B in 4a makes. Additionally, as we see in 4b, the center of the overlap region might

That is, the center of the overlapping region The term of the loss that accounts for mini And once again, authors have no guaranty on the relative orientation of the ellipses, because they don't control it, because they don't even measure it.

Le bail avec les élipses là : c'est joli, mais est ce qu'on a des garanties sur la distance des minimum de chaque tâche ? Parce que donc ça représente le minimum du modèle sur données de task 1 uniquement, et sur données de task 2 uniquement. Mais quid du minimum sur données de task 1 et 2 à la fois (parce qu'au fond, c'est ce qu'on essaye d'atteindre, on peut guère faire mieux nan ?) ?

Dessin avec 4 elipses : l'orientation des elipse est quand même très idéale mdrrrr

NON, plus subtile, curvature : De ce qu'on comprend, la Fisher information est simplement utilisée comme une métrique pour dire que le meilleur chemin pour migrer vers θ_B^* est celui qui nous maintient le plus longtemps dans le minimum pour la tache A

Si on écrit la phrase suivante, on doit être capable d'atteidre SoTA avec juste tradeoffs de modèles entrainés en // : EWC is nothing more than a way to incorporate tradeoffs between tasks in the differentiable optimization process.

Ils ne questionnent pas la légitimité de cette hypothèse : This is with the assumption that there is always an overlapping region for the solution spaces of all tasks for the network to learn them sequentially.

Ils ne contrôlent pas 2 choses : - la distance entre zone acceptable pour tâche A et zone acceptable pour tâche B, donc ils ne contrôlent pas à quel point on va perdre en performance par rapport à ce qu'on pourrait atteindre sur chaque tâche individuellement - ecart entre la valeur du minimum visé par leur méthode de déplacement vers zone acceptable pour B et valeur du minimum atteint pour l'entrainement conjoint sur tâche A et B.

On aimerait certainement entrainer le modèle sur toutes les tâches à la fois, et voir les perfs qu'il obtient, on estime que ça constitue un peu une upper bound de ce qu'on peut atteindre, puisque le continual Learning force une exploration greedy de l'espace des paramètres tandis qu'un entrainement sur toutes les tâches simultanément offre une exploration de l'espace des paramètres plus complet. On veut forcer les paramètres à rester près de θ_A^*, mais les paramètres qui marchent le mieux pour les tâches A et B sont probablement très différents de ceux qui marchent bien pour A seulement.

Avec le dessin des elipses, ils suggèrent très clairement, qu'après s'être entrainés sur la tâche A, on reste à proximité du minimum local sur lequel on a finit l'entrainement. Est-ce que c'est le cas (prcq la loss augmente probablement très fortement quand on passe à la tâche B, donc on est propulser bien loin de notre minimum pour la tâche A, cf notre dessin, et pour que ce ne soit pas le cas, il faudrait fortement diminuer le Learning rate, ce qu'il ne font pas, mais de toute manière ils ne mesurent pas l'augmentation de la loss et n'ont donc aucune chance de controler à quel point ils sont envoyés loin de θ_A^*, et donc, a fortiori, de controler le Learning rate) ? Est-ce qu'on a besoin que ce soit le cas (peut être qu'on ne comprend pas bien comment ça marche, et que l'image n'est là qu'à des fins d'illustration?)

Pourquoi est-ce que c'est un trade-off ? Parce qu'on ne se dirige pas vers un minimum du modèle entrainé sur les données de A et B conjointement, mais vers un minimum du modèle entrainé sur les données de B uniquement, initialié d'une certaine manière, et avec la contrainte qu'on ne veut pas s'éloigner trop des valeurs initiales.

Hum, ya toujours un problème avec les élipses sur lequel on n'a pas encore bien mis le doigt : combien y a t il de loss en jeux ? Et avec quel loss est représenté chaque élément ?

- Elipse A : L_A
- Elipse B : L_B
- Flèche

EWC is good at specializing modules (subset of weights), but it does not tackle the problem of distributing intputs to the modules and merging outputs from the modules. Which are the core challenges of continual Learning. If distributing and merging were not problems, we would simply train separated neural networks, one for each task.

Overcomplexification : In AFEC: Active Forgetting of Negative Transfer in Continual Learning, they introduce a new term in the loss, with a new hyperparameter "while the forgetting factor regulates a penalty to selectively merge the main network parameters with the expanded parameters, so as to learn a better overall representation of both the old tasks and the new task.", so they are trying to mitigate the mitigation made by a regularization method such as EWC. Additionnaly, they hope "to learn a better overall representation of both the old tasks and the new task", which has no reason to happen without enforcing such behavior through constraints, which they don't do.

There is no doubt that EWC is capable of solving benchmarks, but with all these inaccuracies and uncertainties, it certainly does not solve continual learning by "overcoming catastrophic forgetting". At best, it mitigates it, and the purpose of the following section is to attempt to quantify how much EWC is able to mitigate catastrophic forgetting.

10 Methods

10.1 EWC

EWC presents several reproducibility issues. Most paper "reuse" the results of the original paper without reprocuing them Some paper tried to reproduce the result but obtain much lower results Other papers show very inconsistant results And finally, papers trying to apply EWC to new benchmarks show that it is not working well, or even terribly bad. It is likely that these results are significantly poorer because authors don't spend as much energy trying to specificly tune hyperparameters as EWC authors did.

10.2 Our new approach (GroHess or naive)

We will also introduce an obviously bad method (or GroHess) that reaches state of the art on 2 benchmarks.

11 Discussion

11.1 Criticism of the benchmark-oriented approach

We are really good at designing cute benchmarks, like permuted MNIST, which constitute interesting puzzles to solve from the point of view of artificial learning.

but are we good at solving real-world problems ?

11.2 Continual learning usecases

We already argued that continual learning is not a well-defined problem. In this section, we will question the very fundamental assumption of continual learning. and it is not clear what are the usecases of continual learning.

Here is a list each and every usecase of continual learning :

- Adapt to new data quickly https://neptune.ai/blog/continual-learning-methods-and-application
  - Bank fraud (want to adapt quickly to new method)
- A model needs to be personalized https://neptune.ai/blog/continual-learning-methods-and-application
  - Let's say you maintain a document classification pipeline, and each of your many users has slightly different data to be processed—for example, documents with different vocabulary and writing styles. With continual learning, you can use each document to automatically retrain models, gradually adjusting it to the data the user uploads to the system.

Remarks :

- For bank fraud, we don't have enough knowledge to formulate objections.
- Model personalization sounds like finetuining, not continual learning... does it make sense to try to train a single model that we want to use on every client ? Isn't it better to do it in the LoRA fashion, and fine-tune a module that we store with the client's data ?

12 Conclusion

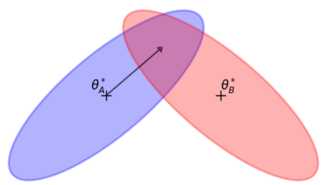
In the end, this efforts put in writting this paper beautifully illustrate Brandolini's law :

> The amount of energy needed to refute bullshit is an order of magnitude bigger than that needed to produce it

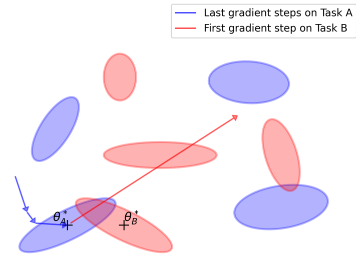
We corrected the work subsequent to two small mistakes, namely, that could have been avoided by a more careful preliminary analysis of the problem. The fact that a method relies on an attempt to balance how much we hurt the learning process on each task naturally introduces trade-offs. These trade-offs are performed through hyperparameters in the loss that set the relative importance of the task, which can easily be mismanipulated or lead to overfitting on the benchmarks at hand.

References

[1] Abhishek Aich. *Elastic Weight Consolidation (EWC): Nuts and Bolts.* 2021. arXiv: 2105.04093.

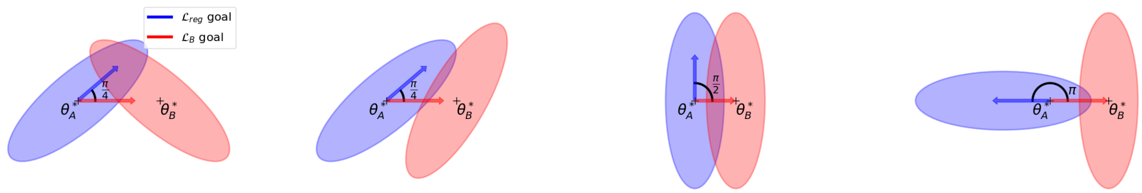


(a) 2 ellipses as depicted in EWC-like papers



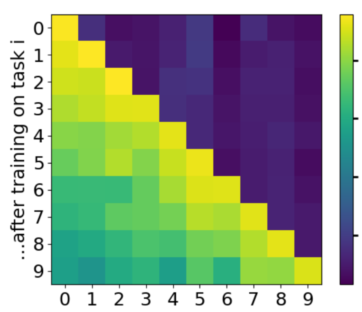
(b) A more realistic view of parameters space

Figure 3: Representations of the parameters space

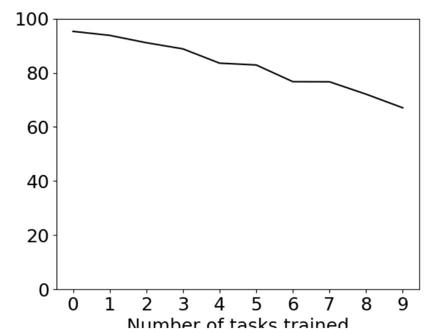


(a) 2 ellipses as depicted in EWC-like papers (b) 2 ellipses as depicted in EWC-like papers (c) 2 ellipses as depicted in EWC-like papers (d) A more realistic view of parameters space

Figure 4: Representations of the parameters space



(a) 2 ellipses as depicted in EWC-like papers



(b) Average test accuracy for tasks trained so far

Figure 5: Representations of the parameters space

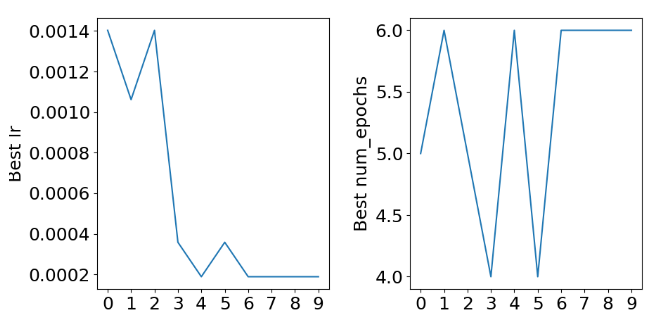


Figure 6: 2 ellipses as depicted in EWC-like papers

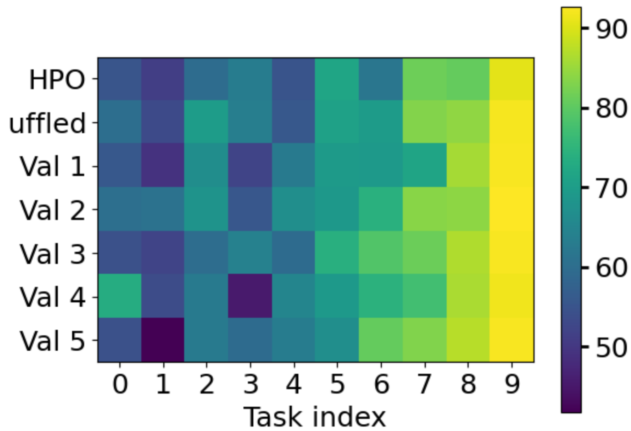
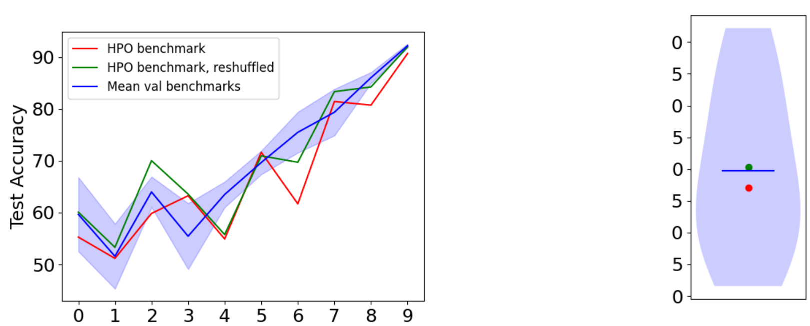


Figure 7: 2 ellipses as depicted in EWC-like papers



(a) 2 ellipses as depicted in EWC-like papers (b) A more realistic view of parameters space

Figure 8: Representations of the parameters space

[2] Thomas Dalgaty et al. “Mosaic: in-memory computing and routing for small-world spike-based neuromorphic systems”. In: *Nature Communications* 15.1 (Jan. 2024). ISSN: 2041-1723. DOI: 10.1038/s41467-023-44365-x. URL: http://dx.doi.org/10.1038/s41467-023-44365-x.

[3] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. *Learning Factored Representations in a Deep Mixture of Experts*. 2014. arXiv: 1312.4314 [cs.LG].

[4] Jason K. Eshraghian et al. *Training Spiking Neural Networks Using Lessons From Deep Learning*. 2023. arXiv: 2109.12894 [cs.NE].

[5] Utku Evci et al. *GradMax: Growing Neural Networks using Gradient Information*. 2022. arXiv: 2201.05125 [cs.LG].

[6] Isobel Claire Gormley and Sylvia Frühwirth-Schnatter. *Mixtures of Experts Models*. 2018. arXiv: 1806.08200 [stat.ME]. URL: https://arxiv.org/abs/1806.08200.

[7] Raia Hadsell et al. “Embracing Change: Continual Learning in Deep Neural Networks”. In: *Trends in Cognitive Sciences* (Nov. 2020). Open Access, Creative Commons Attribution – NonCommercial – NoDerivs (CC BY-NC-ND 4.0). DOI: 10.1016/j.tics.2020.09.004. URL: https://doi.org/10.1016/j.tics.2020.09.004.

[8] Ronald Kemker et al. *Measuring Catastrophic Forgetting in Neural Networks*. 2017. arXiv: 1708.02072.

[9] James Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the National Academy of Sciences* 114.13 (Mar. 2017), pp. 3521–3526. ISSN: 1091-6490. DOI: 10.1073/pnas.1611835114. URL: http://dx.doi.org/10.1073/pnas.1611835114.

[10] Dhireesha Kudithipudi et al. “Biological underpinnings for lifelong learning machines”. In: *Nature Machine Intelligence* 4.3 (Mar. 2022), pp. 196–210. ISSN: 2522-5839. DOI: 10.1038/s42256-022-00452-0. URL: https://doi.org/10.1038/s42256-022-00452-0.

[11] Dhireesha Kudithipudi et al. “Design principles for lifelong learning AI accelerators”. In: *Nature Electronics* 6.11 (Nov. 2023), pp. 807–822. ISSN: 2520-1131. DOI: 10.1038/s41928-023-01054-3. URL: https://doi.org/10.1038/s41928-023-01054-3.

[12] Axel Laborieux et al. “Synaptic metaplasticity in binarized neural networks”. In: *Nature Communications* 12.1 (May 2021), p. 2549. ISSN: 2041-1723. DOI: 10.1038/s41467-021-22768-y. URL: https://doi.org/10.1038/s41467-021-22768-y.

[13] Ziming Liu et al. *Growing Brains: Co-emergence of Anatomical and Functional Modularity in Recurrent Neural Networks*. 2023. arXiv: 2310.07711 [q-bio.NC].

[14] Davide Maltoni and Vincenzo Lomonaco. “Continuous Learning in Single-Incremental-Task Scenarios”. In: *CoRR* abs/1806.08568 (2018). arXiv: 1806.08568. URL: http://arxiv.org/abs/1806.08568.

[15] German Ignacio Parisi et al. “Continual Lifelong Learning with Neural Networks: A Review”. In: *CoRR* abs/1802.07569 (2018). arXiv: 1802.07569. URL: http://arxiv.org/abs/1802.07569.

[16] Javier Lopez Randulfe and Leon Bonde Larsen. *A multi-agent model for growing spiking neural networks*. 2020. arXiv: 2010.15045 [cs.NE].

[17] David Rolnick et al. “Experience Replay for Continual Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/fa7cdfad1a5aaf8370ebeda47a1ff1c3-Paper.pdf.

[18] Hanul Shin et al. “Continual Learning with Deep Generative Replay”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/0efbe98067c6c73dba1250d2beaa81f9-Paper.pdf.

[19] Gido M. van de Ven, Hava T. Siegelmann, and Andreas S. Tolias. “Brain-inspired replay for continual learning with artificial neural networks”. In: *Nature Communications* 11.1 (Aug. 2020), p. 4069. ISSN: 2041-1723. DOI: 10.1038/s41467-020-17866-2. URL: https://doi.org/10.1038/s41467-020-17866-2.

[20] Liyuan Wang et al. *AFEC: Active Forgetting of Negative Transfer in Continual Learning*. 2021. arXiv: 2110.12187.

[21] Xin Yuan, Pedro Savarese, and Michael Maire. *Accelerated Training via Incrementally Growing Neural Networks using Variance Transfer and Learning Rate Adaptation*. 2023. arXiv: 2306.12700 [cs.LG].